ISSN 0254-2757

QI QUAESTIONES INFORMATICAE

Volume 4 Number 3

October 1986

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R10	$ 7	£ 5
Institutions	R15	$14	£10

Circulation and Production

Mr C.S.M. Mueller
Department of Computer Science
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg, 2001

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

A HIGH-LEVEL INTERFACE TO A RELATIONAL DATABASE SYSTEM

S. Berman and L. Walker
University of Cape Town
Rondebosch 7700

A high-level language for accessing a relational database is being developed at UCT. The system, known as HAL (High-level Access Language), provides complete data independence as users view the database in terms of objects and their properties. Constraints on data usage can be incorporated in the data definition and enforced by HAL. In this way a simpler, more controlled interface to a relational database is obtained.

KEY WORDS: relational database, access language, metadata, data independence, subschema

1. INTRODUCTION

Relational databases are easier to use than others primarily because their non-procedurality reduces the navigation problems of the earlier systems. However they still require a thorough knowledge of the relational structure of the data and the correct use of non-trivial operations such as join, which is a form of navigation in that it enables one to pass from one relation to another. The HAL project was initiated to present the programmer with a natural view of data as objects and their properties, where no knowledge of the underlying relations is required. Additional advantages of HAL are enforcing more integrity constraints on data and enabling the metadata (data definition) to be accessed as well as data.

This paper describes the HAL subschema and data manipulation language. Implementation considerations are then outlined and the system evaluated by comparison with conventional relational languages.

2. THE HAL SUBSCHEMA

An example of a simple subschema can be seen in the appendix. A HAL subschema comprises a list of objects optionally followed by a list of tasks. An object is described by several properties each associated with an attribute or relation in the database. The tasks are database manipulating routines which can be called from within programs. Every property is designated as mapping to 1 or M (many) values and has a valuetype which is STRING, INTEGER, REAL, or an object name. The latter case implies that the property references other objects in the database. The application programmer sees only a list of object and property names, and a list of task names.

Constraints can stipulate that certain properties are "compulsory" and cannot be null, or are "id-properties" and cannot have duplicate values. Further, where ever a property of one object references another object (called a "foreign key"), the update and deletion of the foreign object can be specified as nullifying or cascading through to the property, or as being restricted by the property.

"Tasks" [8] are useful for standard operations: it is preferable to store the handling of data once with the data definition, than many times over in programs that use it, possibly inconsistently. A task can have parameters and local variables and can return a result. Tests can be interspersed with executable statements to verify conditions at that particular point. An exception-handling task specified with a test will be invoked whenever the test fails.

3. THE DATA MANIPULATION LANGUAGE

The HAL commands are GET, CREATE, DELETE, UPDATE and NULLIFY. The latter is a special form of UPDATE which allows individual values of a multivalued property to be nullified, leaving other values of that property intact. The user requires no knowledge whatsoever of how the data is stored; he need only know what properties of objects exist. Commands reference data by property names and deal with sets of objects rather than working on an object-by-object basis. So that programmers need not know the ordering of properties within an object, the property, not the object, is the unit of access. Any combination of properties in any order, can be referred to in one statement, and GET can retrieve properties of different kinds of object at the same time. Particular object occurrences to be affected by a command are selected by a predicate expressed in terms of any properties of the object, not only its "key" properties.

The option of file output is provided for extracting large numbers of objects from the database. Arithmetic operations and aggregate functions (average, minimum, maximum, sum, and count) can be applied to database values, and any desired sequencing of objects in a result can be obtained. The concise notation enables multivalued and compound properties, and properties of properties to be referenced easily. Consider the following statement for the HAL subschema in the appendix:

```
CREATE project(name="IBM"; jobhistory.worker.name, jobhistory.hours="Doe",
                36, "Tod", 45, "Fig", 27)
```

Jobhistory is a compound , multivalued property of project which is here being given three values. Each of these comprises a pair of worker-name and hours values, respectively. A "mapping" [6] permits properties of properties to be designated simply: example "jobhistory.worker.name". Special forms of GET are provided to enable metadata to be retrieved. Examples:

```
GET(obj = OBJECT, del = DELETES, edit = UPDATES) ;
```

retrieves all object names with their deletion and update constraints;

```
GET(prop=PROPERTY, of=OBJECTOF, t=TYPE, c=COMPULSORY, u=UNIQUE,
    m=MULTIPLICITY);
```

obtains all property names and descriptions.

The commands are designed to be embedded within a C [7] host language program so their syntax is compatible with that of C. The major difference lies in the fact that C is record-oriented while HAL is set-oriented. This problem is overcome by treating a retrieval statement as a loop control. The (compound) statement following a GET is iterated once for every object retrieved. HAL commands could not be implemented as library routines because of problems with parameter passing for complex predicates. Instead, a preprocessor replaces HAL statements within a C program by appropriate commands manipulating the underlying database. It should be noted that the overhead of such a preprocessor pass through a program exists in most relational database systems.

4. IMPLEMENTATION

The metadata in the HAL subschema is stored in the following relations in the database:

RELATIONS - this stores the relation and attributes in the underlying database as well as their data type

OBJECTS - here there are tuples for each object type and every relation in which that object is referenced. If the object is a foreign key the restrict/cascade/nullify information is recorded along with the name of the foreign key attribute

PROPERTIES - this gives the attribute or relation representing each property. If a relation, the property is termed "complex" (eg. jobhistory and workdone in the appendix.) The objecttype and id columns in PROPERTIES indicate foreign keys. In the appendix "worker" and "operation" are two such properties mapped to work.eno (specifying workers via their employee number) and work.jno (designating a project number), respectively.

JOINS - this gives the attributes to match when joining one relation to another for a complex property.

The appendix illustrates the metadata created for a simple subschema.

The HAL preprocessor parses commands, checks them against the stored metadata and translates them into equivalent operations on the relational database. When a task is invoked the preprocessor extracts any condition associated with the call in the program, translates it into an equivalent predicate on the underlying database and appends it to all database manipulation commands in the task. The system is being tested on an Ingres database and hence HAL statements are translated to EQUEL [9]. A HAL command generally requires several EQUEL operations, since what appears to the programmer as a single object is usually represented by several relations. The preprocessor deduces that a join is required when it encounters a foreign or complex property.

Consider firstly the handling of a foreign key such as worker. Its PROPERTIES entry equates it with work.eno, and its "objecttype" and "id" indicate that it references employee objects by number. Employee number is found in PROPERTIES to be represented by emp.eno. Thus the join clause "WHERE work.eno = emp.eno" is used. For a complex property, PROPERTIES shows that it is represented by an entire relation. The JOINS tuple for that property is thus examined to obtain the join clause.

Handling CREATE, NULLIFY and UPDATE commands requires that properties be re-ordered before equivalent EQUEL statements are devised. In this way properties in one relation are treated, then those in the next relation, and so on. When the PROPERTIES entry indicates that a complex property is being inserted, the metadata for the corresponding relation is examined to determine its compulsory properties. Unfortunately, the foreign key value may be given under some other property name, eg:

```
CREATE project(number=102; name="NCR"; jobhistory.hours,
               jobhistory.worker.name=12, "Doe", 24, "Fig", 9 "Poe");
```

The complex property jobhistory is represented by the work relation. Its attribute work.jno represents "operation", a compulsory foreign key. The statement did indeed indicate a particular operation giving the project number 102, but has understandably not repeated it in the form jobhistory.operation.number. To deal with this, the missing property is determined from "id" in PROPERTIES (here, project number). This property value is therefore sought in the statement if the expected foreign key is not explicitly given.

Another difficulty arises from indirect identification of the foreign key, eg:

```
CREATE activity(worker.name, hours, operation.number="Doe", 12, 24, "fig",
                27, 18, "Poe", 33, 3);
```

Activity is represented by the work relation. Its worker property is a foreign key represented as employee numbers. However, the command above has identified employees by their name. The system detects that name and not number has been supplied, and first retrieves the employee number corresponding to the given name. The EQUEL statements to handle insertion of the first activity are thus:

```
RANGE OF empx IS emp
RETRIEVE (nvar[0] = empx.eno, flag = ANY(empx.eno where empx.ename = "Doe"))
      WHERE empx.ename = "Doe"
If (flag == 1) APPEND TO work (eno = nvar[0], hrs = 12, jno = 24)
```

The HAL system has to impose all subschema constraints. To enforce restricted updates and deletions, a RETRIEVE ANY is used to detect tuples preventing the desired operation. Additional EQUEL statements are created to propagate changes and deletions through the database. Finally, enforcing uniqueness of id-properties involves using ANY to detect duplicates.

5. EVALUATION

The major advantages of HAL over existing relational languages are that several underlying relations can be referenced in one statement as if the information were all stored together, and that objects can be uniquely distinguished via any identifier. There is no need to distinguish "attributes" from "associations" even though they may be implemented differently and the join operation of their relational algebra is replaced by the simple concept of a mapping. In addition HAL can restrict data removal and alteration where appropriate, detect duplicates or null values where they are not allowed, and propagate deletions and updates through the database. The metadata relations created by the system are accessible to programmers, enabling them to query the data definition. Figure 1 illustrates the relative simplicity of HAL compared with the conventional database languages: relational algebra[5], relational calculus[4], EQUEL[9] and SQL[1]. Query: Give the names of all employees who have worked on the "ICL" project (for the database in the appendix).
Phrasing the same query in HAL:

```
GET (result=jobhistory.worker.name) WHERE project.name == "ICL";
```

Phrasing the sample query in the relational algebra:

```
JOIN emp AND work OVER eno GIVING templ
JOIN templ AND job OVER jno GIVING temp2
SELECT temp2 WHERE jname = "ICL" GIVING temp3
PROJECT temp3 OVER ename GIVING result
```

Phrasing the sample query in the relational calculus:

```
RANGE wx work
RANGE jx job
GET result(emp.ename): ∃wx ∃jx(emp.eno = wx.eno & wx.jno = jx.jno &
                                 jx.jname = "ICL")
```

Phrasing the sample query in EQUEL:

```
RANGE OF ex IS emp
RANGE OF jx IS job
RANGE OF wx IS work
RETRIEVE (result=ex.ename) WHERE ex.eno=wx.eno and wx.jno=jx.jno and
                                 jx.jname = "ICL"
```

Phrasing the sample query in SQL:

```
SELECT ename
FROM emp
WHERE eno IN (SELECT eno
             FROM work
             WHERE jno IN (SELECT jno
                           FROM job
                           WHERE jname = "ICL"))
```

figure 1
Comparison of Five Relational Languages

HAL has demonstrated that a high-level interface to a relational database can provide complete data independence, thus enhancing ease of use and making programs immune to changes in the schema. The major problem remaining is that the data manipulation commands are embedded in a foreign host programming language. A persistent programming language based on HAL is accordingly being developed [2].

REFERENCES

1. Astrahan M.M. and Chamberlin D.D. [1975], Implementation of a Standard English Query Language, *Communications of the ACM*, **18**.
2. Berman S. [1986], PPL - A Preliminary Report, *Quæstiones Informaticæ*, **4**, 25-32.
3. Codd E.F. [1970], A Relational System of Data for Large Shared Data Banks, *Communications of the ACM*,**13**.
4. Codd E.F. [1971], A Data Base Sublanguage Founded on Relational Calculus, *Proc . ACM SIGFIDET Workshop on Data Description, Access and Control.*
5. Codd E.F. [1972], Relational Completeness of Data Base Sublanguages, *Data Base Systems*, Courant Computer Science Symposia Series (6), Prentice-Hall.
6. Hammer M. and McLeod D. [1981], Database Description with SDM: A Semantic Database Model, *ACM Transactions on Database Systems*, **6**, 351-386.
7. Kernighan B.W. and Ritchie D.M. [1978], *The C Programming Language*, Prentice-Hall.
8. Mylopoulos J., Bernstein P.A. and Wong H.K.T. [1980], A Language Facility for Designing Database-Intensive Applications, *ACM Transactions on Database Systems*, **5**, 185-207.
9. NCR Corp. [1983], *TOWER Database Manager*, Relational Technology Inc.

APPENDIX - SIMPLE EXAMPLE

Employee-Work Database Relations

```
EMP(ename,eno,wage)
JOB(jname,jno,charge)
WORK(jno,eno,hrs)
```

Subschema for Employee-Work Database (The TRANSLATION section is transparent to application programmers)

```
OBJECT Employee
 ID-PROPERTY
 Name                           1 STRING compulsory
 Number                         1 INTEGER unique compulsory
PROPERTIES
 Wage                           1 REAL
 Workdone                       M Activity update cascades
OBJECT Project
 ID-PROPERTY
 Name                           1 STRING unique compulsory
 Number                         1 INTEGER unique
PROPERTIES
 Charge                         1 REAL
 Jobhistory                     M Activity update cascades
OBJECT Activity
 ID-PROPERTY
 Operation                      1 Project compulsory delete restricted update cascades
 Worker                         1 Employee delete nullifies update cascades
PROPERTIES
 Hours                          1 REAL

TRANSLATION
 OBJECT Employee
 Name                           EMP.ename
 Number                         EMP.eno
 Wage                           EMP.wage
 Workdone                       WORK
 OBJECT Project
 Name                           JOB.jname
 Number                         JOB.jno
 Charge                         JOB.charge
 Jobhistory                     WORK
```

```
OBJECT Activity
Operation                  WORK.jno = Project.Number
Worker                     WORK.eno = Employee.Number
Hours                      WORK.hrs
```

Metadata for above subschema

RELATIONS

relation	column	format
EMP	eno	int
EMP	ename	char
EMP	wage	int
JOB	jno	int
JOB	jname	char
JOB	charge	int
WORK	eno	int
WORK	jno	int
WORK	hrs	int

OBJECTS

objectname	relation	column	deletes	updates	prop
employee	emp				
employee	work	eno	nullify	cascade	number
project	job				
project	work	jno	restrict	cascade	number
activity	work				

PROPERTIES

propname	object	relation	column	objecttype	id	c	u	m
name	employee	emp	ename			Y	N	1
number	employee	emp	eno			Y	Y	1
wage	employee	emp	wage			N	N	1
workdone	employee	work		activity		N	N	M
name	project	job	jname			Y	Y	1
number	project	job	jno			N	Y	1
charge	project	job	charge			N	N	1
jobhistory	project	work		activity		N	N	M
worker	activity	work	eno	employee	number	N	N	1
ııııııııı	activity	work	jno	project	number	Y	N	1
hours	activity	work	hrs			N	N	1

(in the above relation the names "compulsory", "multiplicity" and "unique" have been abbreviated to "c", "m" and "u" respectively).

JOINS

property	relation1	column1	relation2	column2
jobhistory	job	jno	work	jno
workdone	emp	eno	work	eno

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review artilces and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Prof. G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0106
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

1. Ashcroft E. and Manna Z., The Translation of 'GOTO' Programs to 'WHILE' programs., *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255, 1972.
2. Bohm C. and Jacopini G., Flow Diagrams, Turing Machines and Languages with only Two Formation Rules., *Comm. ACM*, **9**, 366-371, 1966.
3. Ginsburg S., Mathematical Theory of Context-free Languages, McGraw Hill, NewYork, 1966.

Proofs and reprints

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.